Compiler Design

Project: ChatterLang

Language Overview:

ChatterLang is a unique, domain-specific programming language (DSL) designed to simplify the creation of chatbot workflows, event-driven systems, and conversational AI applications. It combines high-level syntax for natural language processing tasks with low-level control over chatbot behavior, making it both developer-friendly and powerful.

Lexical Structure

Token Types

1. Keywords: Reserved words that define the structure of the language.
    - Examples: Flow, Bot, User, If, Else, End, Let
2. Identifiers: Variable names enclosed in curly braces {}.
    - Examples: {name}, {age}, {response}.
3. Literals: Constant values.
    - String: Text enclosed in double quotes, e.g., "Hello!".
    - Number: Integers or floating-point numbers, e.g., 123, 3.14.
4. Operators:
    - Arithmetic: +, -, *, /.
    - Logical: &&, ||, !.
    - Comparison: ==, !=, , , =, =.
5. Separators:
    - {, }, :, [,], ,, (,).
6. Whitespace: Ignored except to separate tokens.
7. Comments: Start with # and extend to the end of the line.

Grammer:

Program: ⟶ Flow Program | ε
Flow: ⟶ "Flow" Identifier ":" Statements "End"
Statements: ⟶ Statement Statements | ε
Statement: ⟶ Bot_Statement | User_Statement | If_Statement | Loop_Statement
| Assignment
Bot_Statement: ⟶ "Bot:" String
User_Statement: ⟶ "User:" Identifier
If_Statement: ⟶ "If" Condition ":" Statements Else_Clause "End"
Else_Clause: ⟶ "Else:" Statements | ε
Loop_Statement: ⟶ "Loop:" Statements "End"
Assignment: ⟶ "Let" Identifier "=" Condition
Condition: ⟶ Value Expression
Expression: ⟶ Operator Value Expression | ε
Value: ⟶ String | Number | Identifier
String: ⟶ "\"" .*? "\""
Number: ⟶ [0-9]+[0-9]*
Identifier: ⟶ "{" [a-zA-Z_][a-zA-Z0-9_]* "}"
Operator: ⟶ "==" | "!=" | "<" | ">" | "<=" | ">=" | "=" | "%" | "\+" | "-" | "*" | "/"

Symbol Table:

Non-Terminal	Unique Character	Description
Program	P	The starting rule of the program, consisting of one or more flows.
Flow	F	Represents a flow, defined with "Flow ... End".
Statements	S	A sequence of one or more statements within a flow or block.
Statement	T	Represents a single statement (bot, user, if, loop, assignment).
Bot_statement	B	A statement where the bot sends a message to the user.
User_statement	U	A statement where the user provides input.
If_statement	I	A conditional statement with an optional "Else" block.
Else_clause	E	Represents the "Else" block in an If statement (optional).
Loop_statement	L	A loop block where statements are repeated.
Assignment	A	A variable assignment where a value is stored in an identifier.
Condition	C	Represents a logical comparison between two values.
Expression	EXP	Handles recursive parsing of operators and values for complex expressions.
Value	V	Any value: a string, number, or identifier.
String	Q	A string literal enclosed in double quotes ("...").
Number	N	A numeric literal, either an integer or a floating-point number.
Identifier	ID	A variable name enclosed in curly braces ({}), e.g., {name}.
Operator	O	Represents any arithmetic or logical operator (e.g., ==, !=, +, -).

Simplified Grammar:

P: ⟶ F P | ε
F: ⟶ Flow ID : S End
S: ⟶ T S | ε
T: ⟶ B | U | I | L | A
B: ⟶ Bot: Q
U: ⟶ User: ID
I: ⟶ If C : S E end
E: ⟶ Else: S | ε
L: ⟶ Loop: S end
A: ⟶ Let ID = C
C: ⟶ V EXP
EXP: ⟶ O V EXP | ε
V: ⟶ Q | N | ID
Q: ⟶ “String”
N: ⟶ [0-9]+[0-9]*
ID: ⟶ { [a-zA-Z_][a-zA-Z0-9_]* }
O: ⟶ Operators

Sample Code:

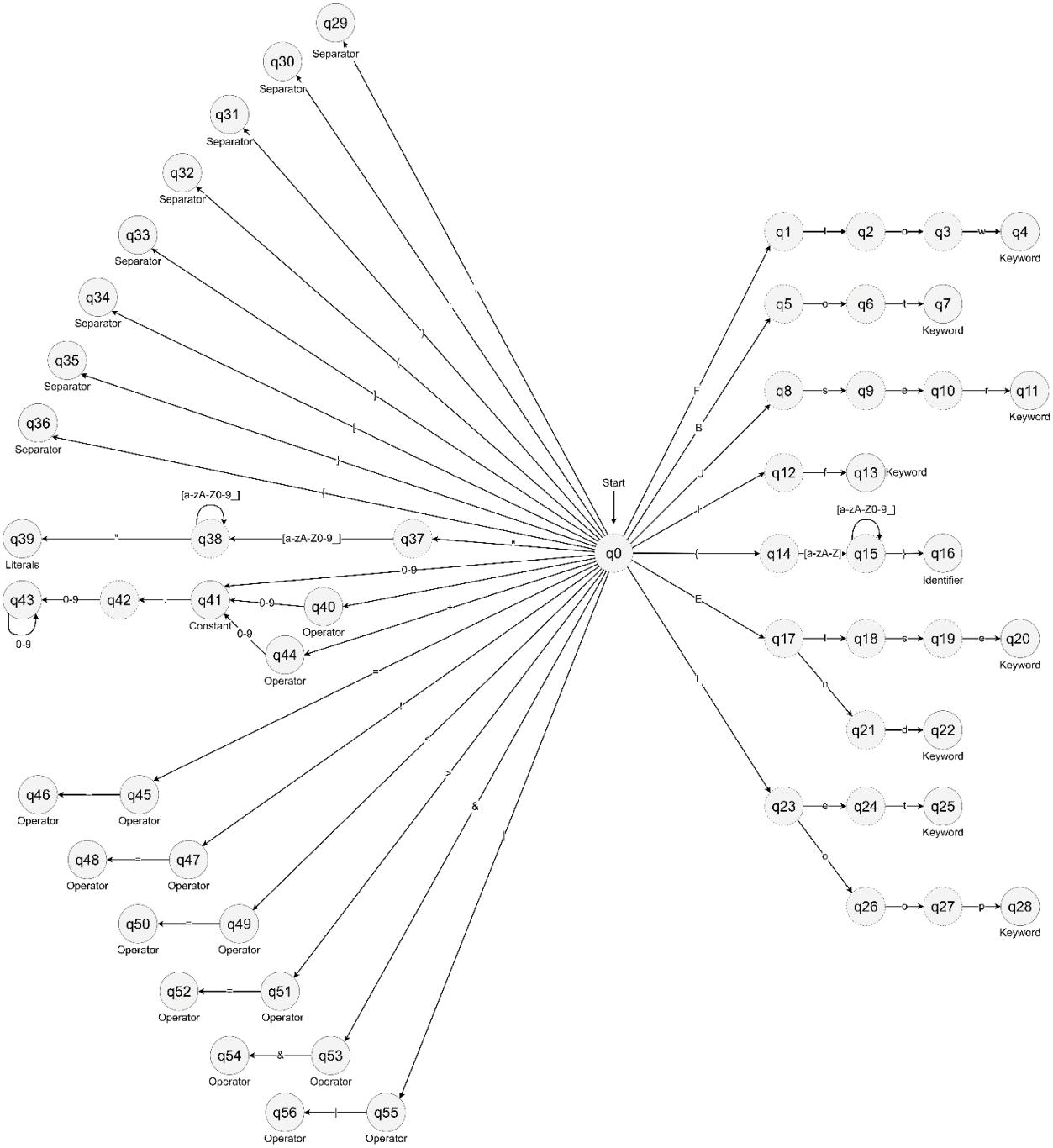
```
Flow {GreetUser}:
   Bot: "Hello! What's your name?"
   User: {name}
   Bot: "Nice to meet you, {name}!"

   If {name} == "Alice":
      Bot: "You're an admin. Welcome back!"
   Else:
      Bot: "Hi there, {name}!"
   End

   Loop:
      Bot: "Do you want to continue? (yes/no)"
      User: {response}
      If {response} == "no":
         Bot: "Goodbye!"
         End
      End
   End
End
```

Lexer:

1. **DFA:**


q0
Start
q1
q2
q3
q4
Keyword
q5
q6
q7
Keyword
q8
q9
q10
q11
Keyword
q12
q13
Keyword
q14
q15
q16
Identifier
[a-zA-Z0-9_]
[a-zA-Z]
q17
q18
q19
q20
Keyword
q21
q22
Keyword
q23
q24
q25
Keyword
q26
q27
q28
Keyword
F
B
U
I
{
E
L
q29
Separator
q30
Separator
q31
Separator
q32
Separator
q33
Separator
q34
Separator
q35
Separator
q36
Separator
)
(
]
[
}
{
q37
q38
q39
Literals
[a-zA-Z0-9_]
"
"
0-9
q40
Operator
+
q41
Constant
0-9
q42
.
0-9
q43
0-9
0-9
q44
Operator
=
!
<
>
&
|
q45
Operator
q46
Operator
=
q47
Operator
q48
Operator
=
q49
Operator
q50
Operator
=
q51
Operator
q52
Operator
=
q53
Operator
q54
Operator
&
q55
Operator
|
q56
Operator

2. **Sample Code:**

```
Flow {GreetUser}:
    Bot: "Hello, what's your name?"
    User: {name}
    Bot: "Nice to meet you, {name}!"
    Let {counter} = 10
End
```

3. **Tokens:**

Keyword	Flow
Identifier	{GreetUseer}
Separator	:
Keyword	Bot
Separator	:
String	“Hello, what’s your name?”
Keyword	User
Separator	:
Identifier	{name}
Keyword	Bot
Separator	:
String	“Nice to meet you, {name}!”
Keyword	Let
Identifier	{counter}
Operator	=
Number	10
Keyword	End
EOF	$

LL(1) Parsing:

First and Follow:

Non-Terminal	First	Follow
P	{ Flow, ε }	{ $ }
F	{ Flow }	{ $, Flow }
S	{ Bot:, User:, If, Loop:, Let, ε }	{ End, Else:, $ }
T	{ Bot:, User:, If, Loop:, Let }	{ Flow, Bot:, User:, If, Loop:, Let,$}
B	{ Bot: }	{ Flow, Bot:, User:, If, Loop:, Let,$}
U	{ User: }	{ Flow, Bot:, User:, If, Loop:, Let,$}
I	{ If }	{ Flow, Bot:, User:, If, Loop:, Let,$}
E	{ Else:, ε }	{ End }
L	{ Loop: }	{ Flow, Bot:, User:, If, Loop:, Let,$}
A	{ Let }	{ Flow, Bot:, User:, If, Loop:, Let, $}
C	{ {, [0-9]+[0-9]*, "}	{ :, Flow, Bot:, User:, If, Loop:, Let,$ }
EXP	{ OPERATOR, ε }	{ :, Flow, Bot:, User:, If, Loop:, Let,$}
V	{{, [0-9]+[0-9]*, ""}	{ OPERATOR, $ }
Q	{ " }	{ Flow, Bot:, User:, If, Loop:, Let, operators, $}
N	{ [0-9]+[0-9]* }	{ OPERATOR , $}
ID	{ { }	{ :, ε, Flow, Bot:, User:, If, Loop:, Let, =, operators }
O	{ OPERATOR }	{ ", number, { }, $ }

Parsing Table:

	Flow	:	End:	Bot:	User:	If	Else:	Loop:	Let	=	“, String	{, Identifier	[0-9], Number	”	}	Operator	$
P	P: ~> F P																P: ~> ε
F	F: ~> Flow ID : S End																
S			S: ~> ε	S: ~> T S	S: ~> T S	S: ~> T S	S: ~> ε	S: ~> T S	S: ~> T S								S: ~> ε
T				T: ~> B	T: ~> U	T: ~> I		T: ~> L	T: ~> A								
B				B: ~> Bot: Q													
U					U: ~> User: ID												
I						I: ~> If C : S E end											
E			E: ~> ε				E: ~> Else: S										
L								L: ~> Loop: S end									
A									A: ~> Let ID = C								
C											C: ~> V EXP	C: ~> V EXP	C: ~> V EXP				
EXP	EXP: ~> ε	EXP: ~> ε		EXP: ~> ε	EXP: ~> ε	EXP: ~> ε		EXP: ~> ε	EXP: ~> ε							EXP: ~> O V EXP	EXP: ~> ε
V											V: ~> Q	V: ~> ID	V: ~> N				
Q											Q: ~> ”String "						
N													N: ~> [Number]				
ID												ID: ~> { Identifier }					
O																O: ~> Operators	

Input String:

Flow { greet } : Bot: "Hello" User: { name } End $

Parsing Table:

Step	Stack	Input	Action
1	P $	Flow { greet } : Bot: "Hello" User: { name } End $	Expand P -> F P
2	F P $	Flow { greet } : Bot: "Hello" User: { name } End $	Expand F -> Flow D : S End
3	Flow D : S End P $	Flow { greet } : Bot: "Hello" User: { name } End $	Match Flow and consume input
4	D : S End P $	{ greet } : Bot: "Hello" User: { name } End $	Expand D -> { identifier }
5	{ identifier } : S End P $	{ greet } : Bot: "Hello" User: { name } End $	Match { and consume input
6	identifier } : S End P $	greet } : Bot: "Hello" User: { name } End $	Match identifier and consume
7	} : S End P $	} : Bot: "Hello" User: { name } End $	Match } and consume input
8	: S End P $	: Bot: "Hello" User: { name } End $	Match : and consume input
9	S End P $	Bot: "Hello" User: { name } End $	Expand S -> T S
10	T S End P $	Bot: "Hello" User: { name } End $	Expand T -> B
11	B S End P $	Bot: "Hello" User: { name } End $	Expand B -> Bot: Q
12	Bot: Q S End P $	Bot: "Hello" User: { name } End $	Match Bot: and consume input
13	Q S End P $	"Hello" User: { name } End $	Expand Q -> " .*? "
14	" .*? " S End P $	"Hello" User: { name } End $	Match "Hello" and consume
15	S End P $	User: { name } End $	Expand S -> T S
16	T S End P $	User: { name } End $	Expand T -> U
17	U S End P $	User: { name } End $	Expand U -> User: D
18	User: D S End P $	User: { name } End $	Match User: and consume input
19	D S End P $	{ name } End $	Expand D -> { identifier }
20	{ identifier } S End P $	{ name } End $	Match { and consume input
21	identifier } S End P $	name } End $	Match identifier and consume
22	} S End P $	} End $	Match } and consume input
23	S End P $	End $	Expand S -> ε
24	End P $	End $	Match End and consume input
25	P $	$	Expand P -> ε
26	$	$	Accept: Parsing successful!